軽量型情報 LSP の開発 — type-guessr

Ruby Association Activity Report 2026/08/04

SANGYONG SIM

Self Introduction

SIM SANGYONG (@shia)

- STORES Inc.
- GitHub: @riseshia
- X: @riseshia

Motivation

I don’t want to write types in Ruby.

but sometimes I’m happy to get type information.

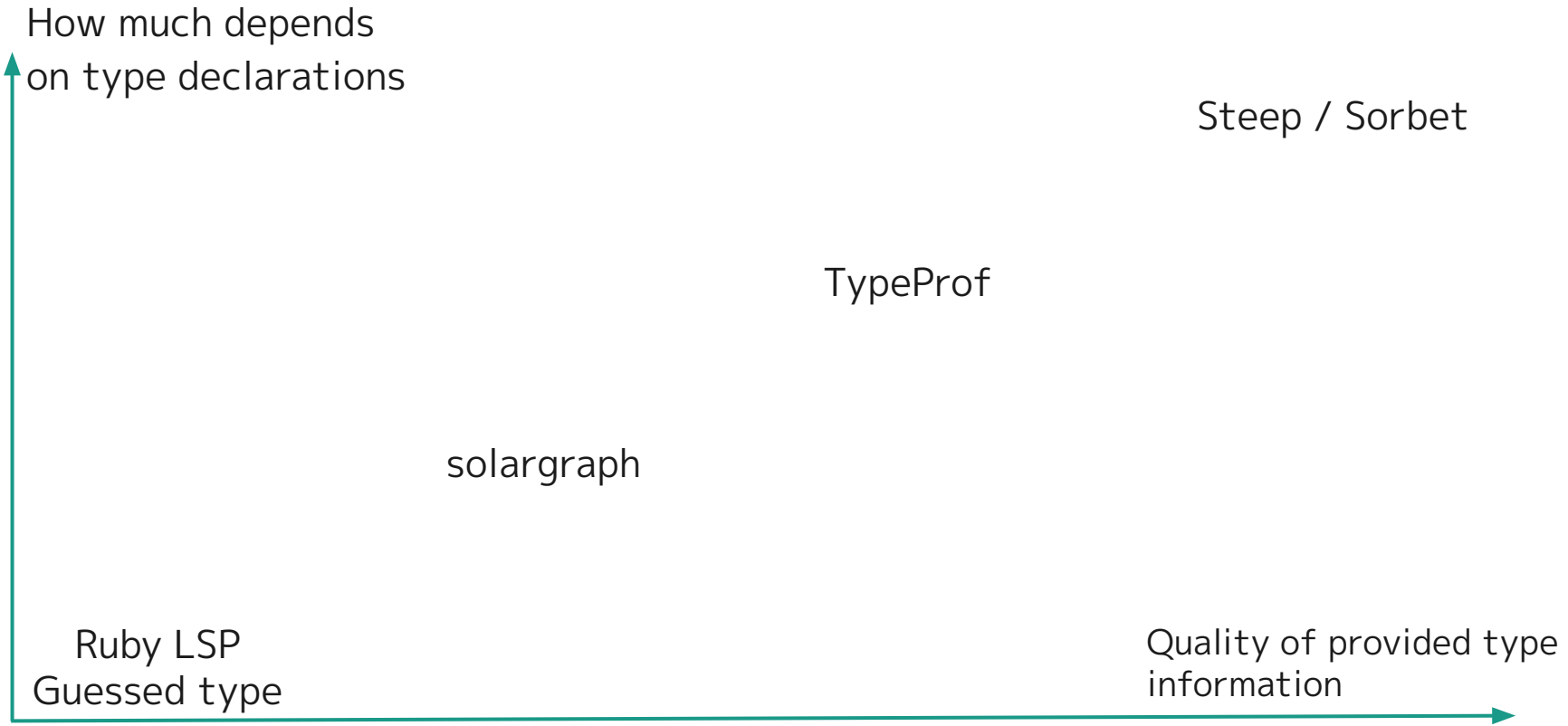
Like this

```
active_user = User.where(active: true)
member = User.where.not(role: :admin)


first_member = member.first
```

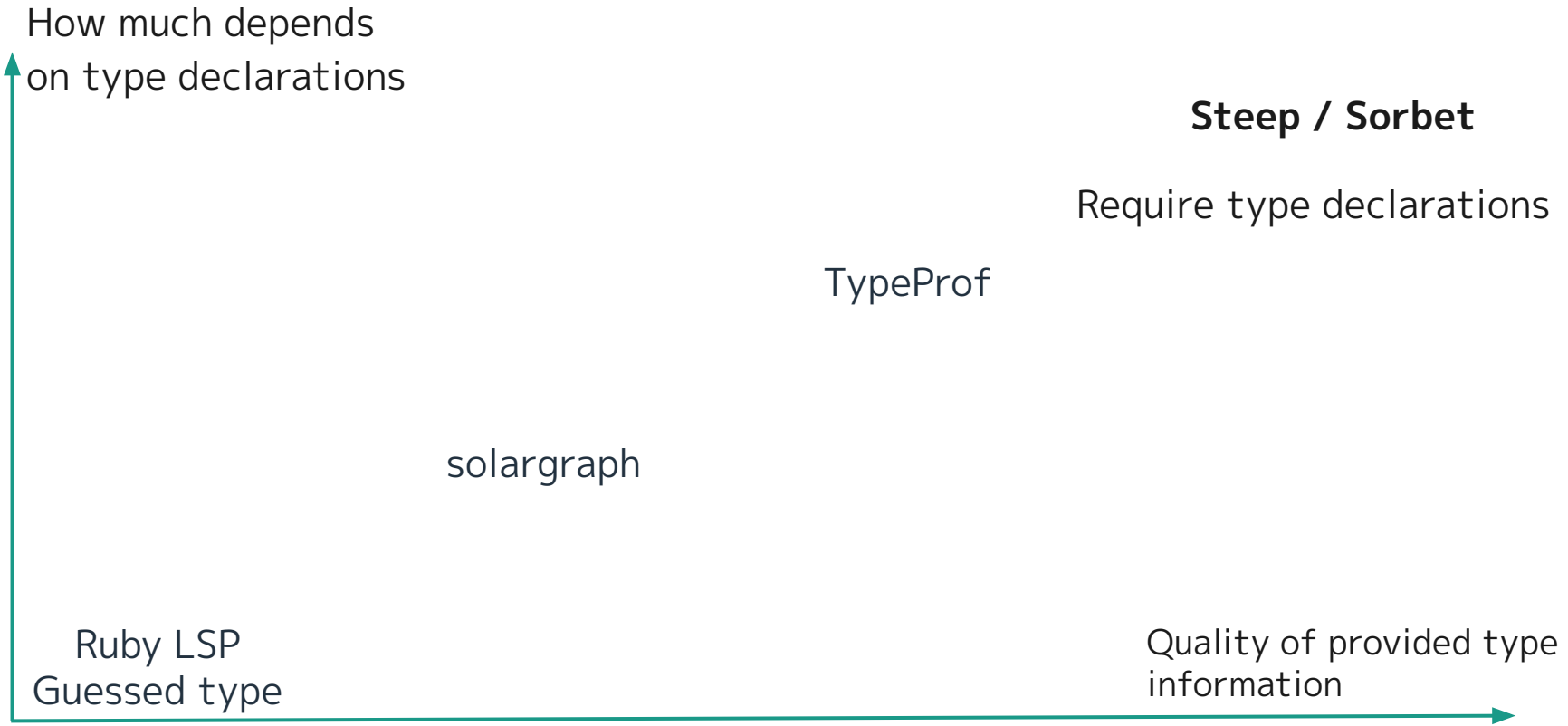
Like this


active_user = User.where(active: true)
member = User.where.not(role: :admin)
Guessed Type: User?
first_member = member.first

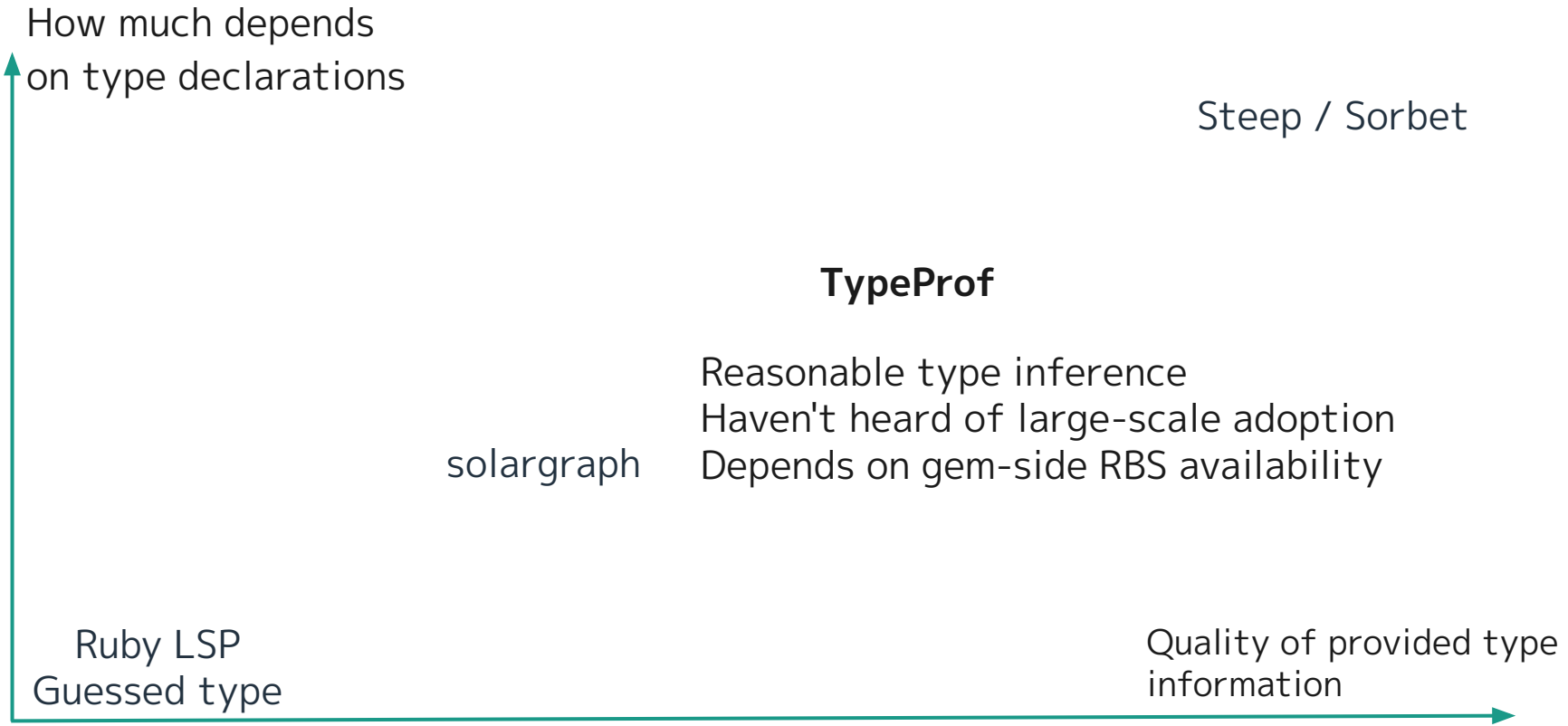
Landscape of type support tools for Ruby



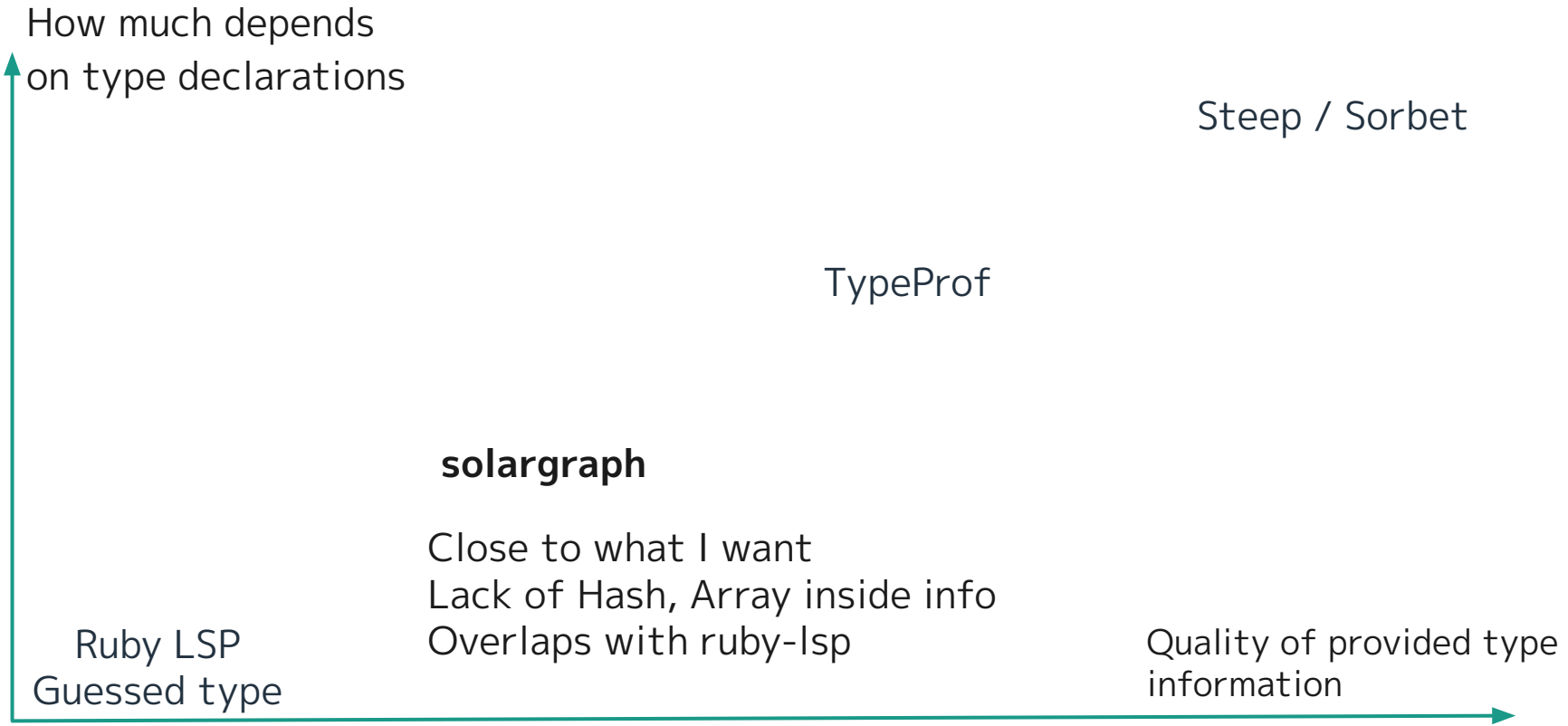
Landscape of type support tools for Ruby



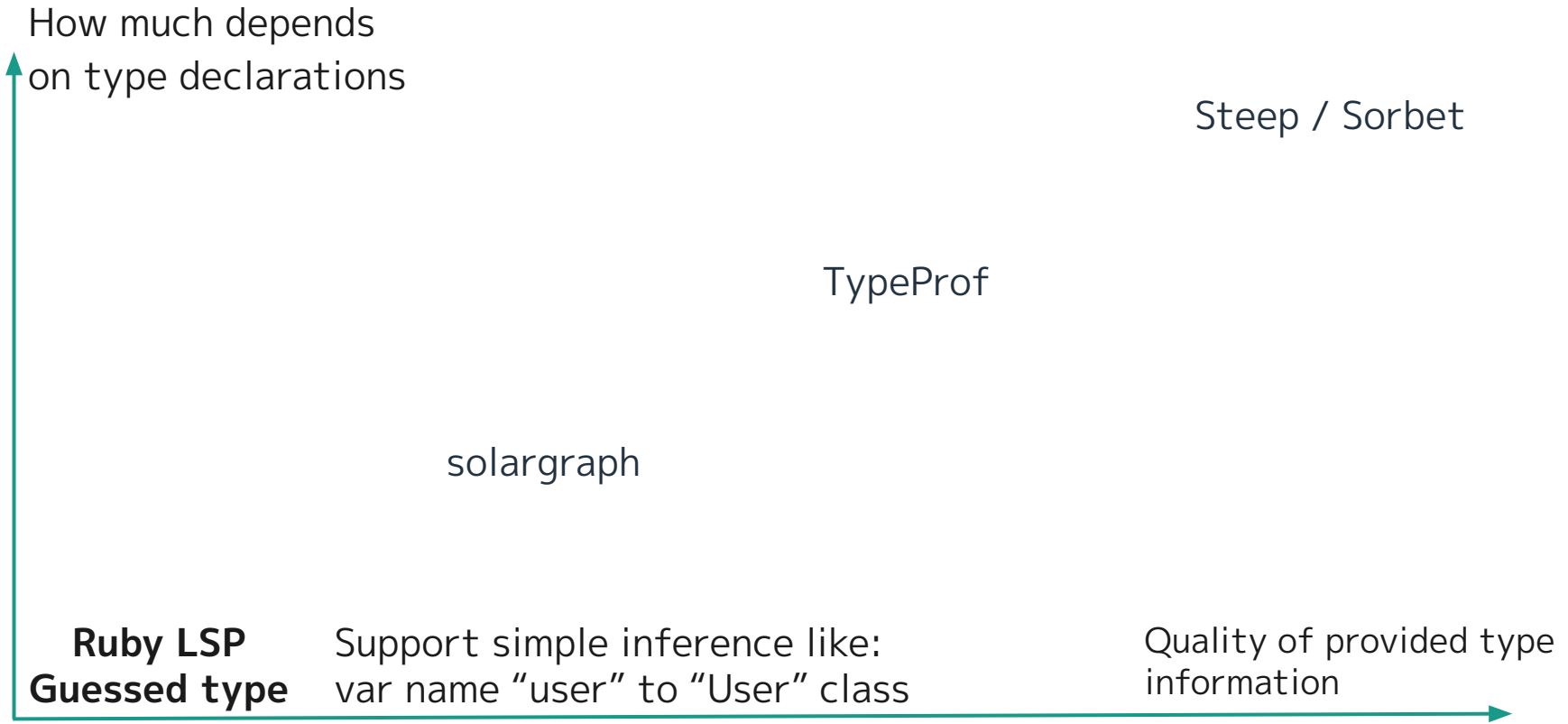
Landscape of type support tools for Ruby



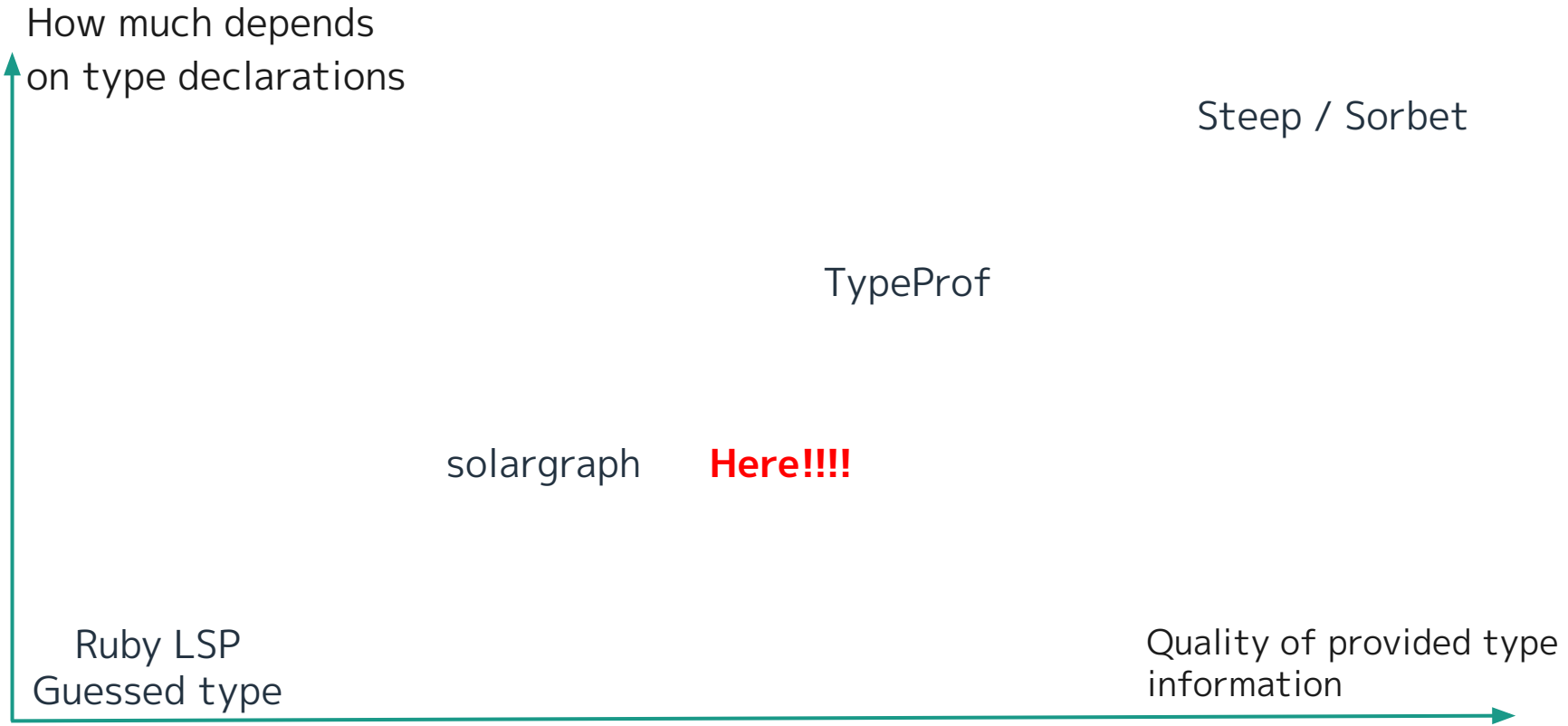
Landscape of type support tools for Ruby



Landscape of type support tools for Ruby



Landscape of type support tools for Ruby



What I want

Sacrifice a bit of accuracy to efficiently produce plausible type info?

Proposal

- Goal: Impl LSP Server to provide practical type information
- How: Via heuristic that sacrifice a bit of accuracy to gain more coverage.

type-guessr

A Ruby LSP addon that provides heuristic type inference to enhance IDE features like Hover and Go to Definition.

Topics

- How to infer type of code
- How to get more type information from code
- How expressive is the type
- How to support DSL-like code

Approach

Duck typing:

"If it can quack, treat it as a duck"

- Ignore the type. Only behavior (methods) matters
- behavior as a type

Approach - Duck typing inspired heuristic

What we will do:

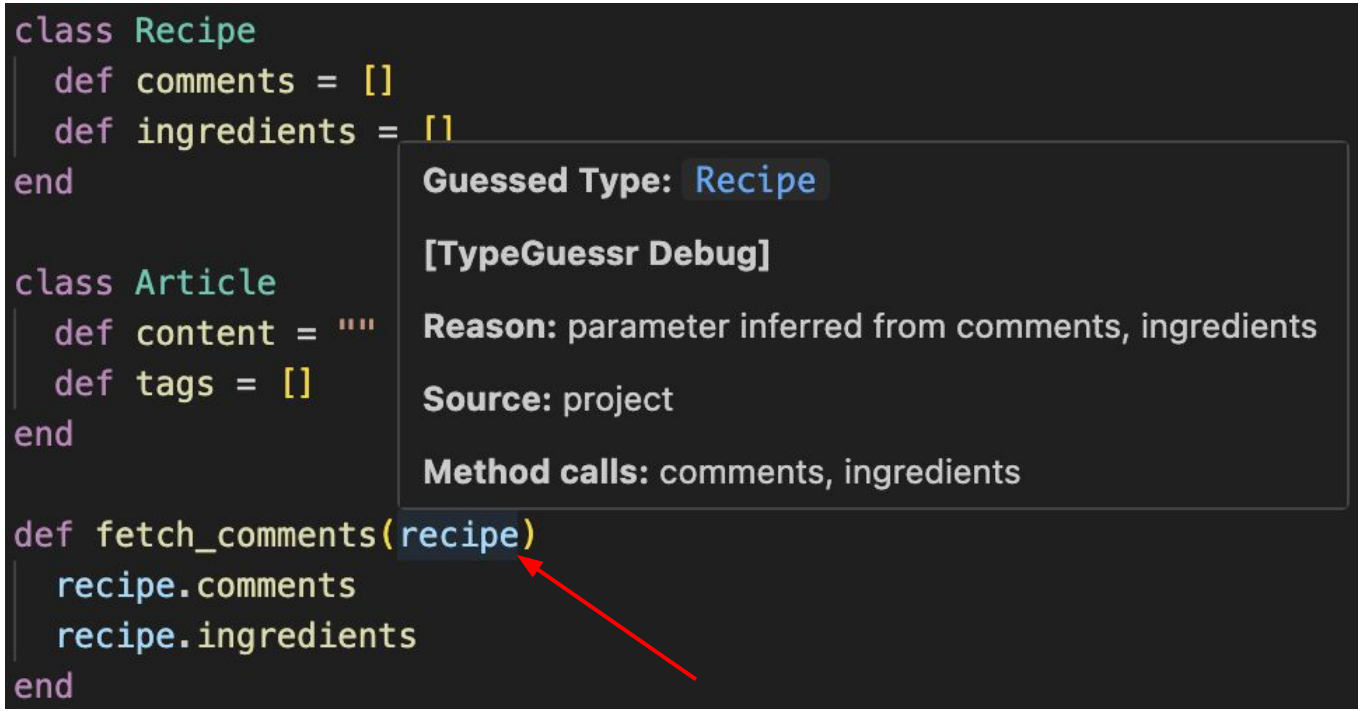
"quack() is called on this object. The only class with this is… Duck class"

- **Infer the type from observed behavior.**
- **type from behavior.**

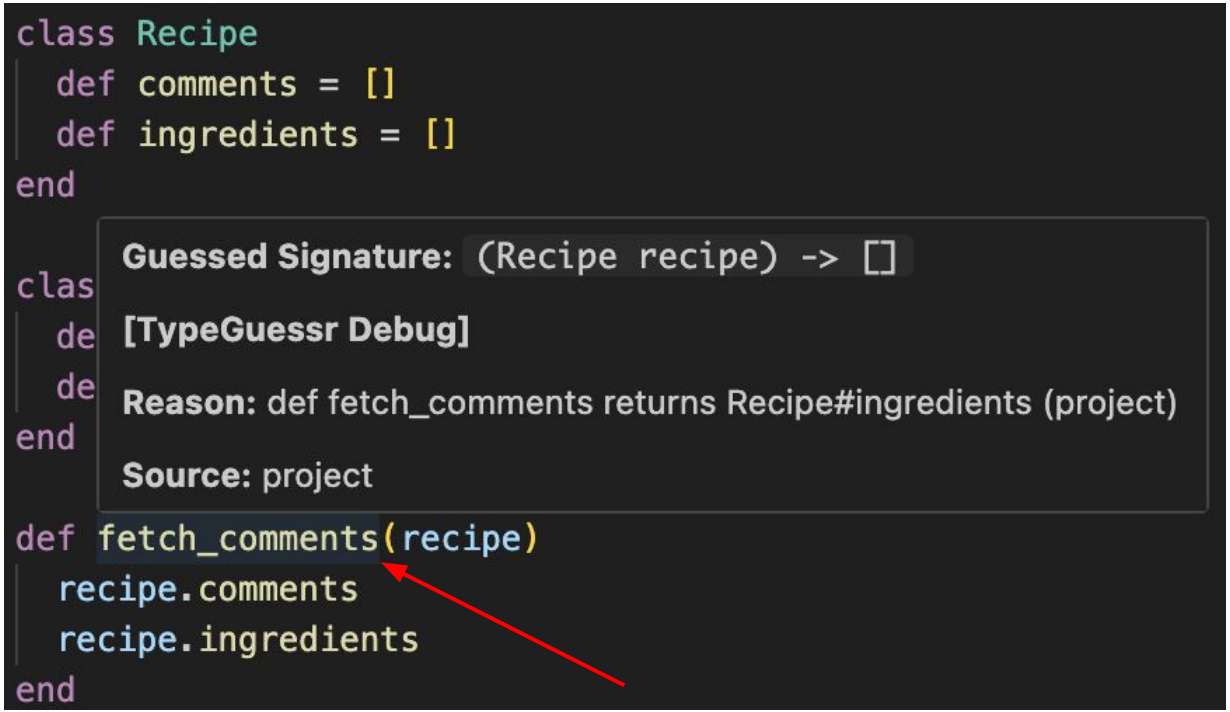
Approach - Duck typing inspired heuristic

```
class Recipe
  def comments = []
  def ingredients = []
end

class Article
  def content = ""
  def tags = []
end

def fetch_comments(recipe)
  recipe.comments
  recipe.ingredients
end
```

Approach - Duck typing inspired heuristic

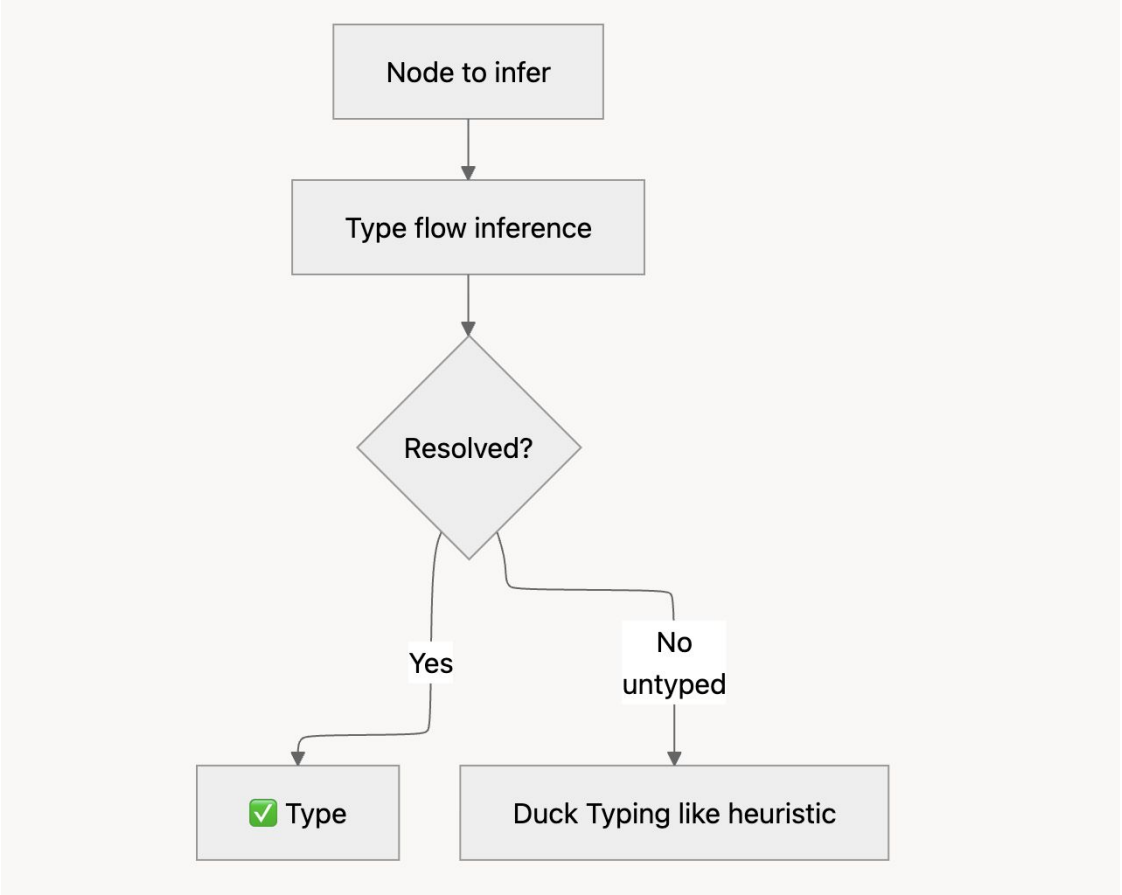


Approach - Duck typing inspired heuristic

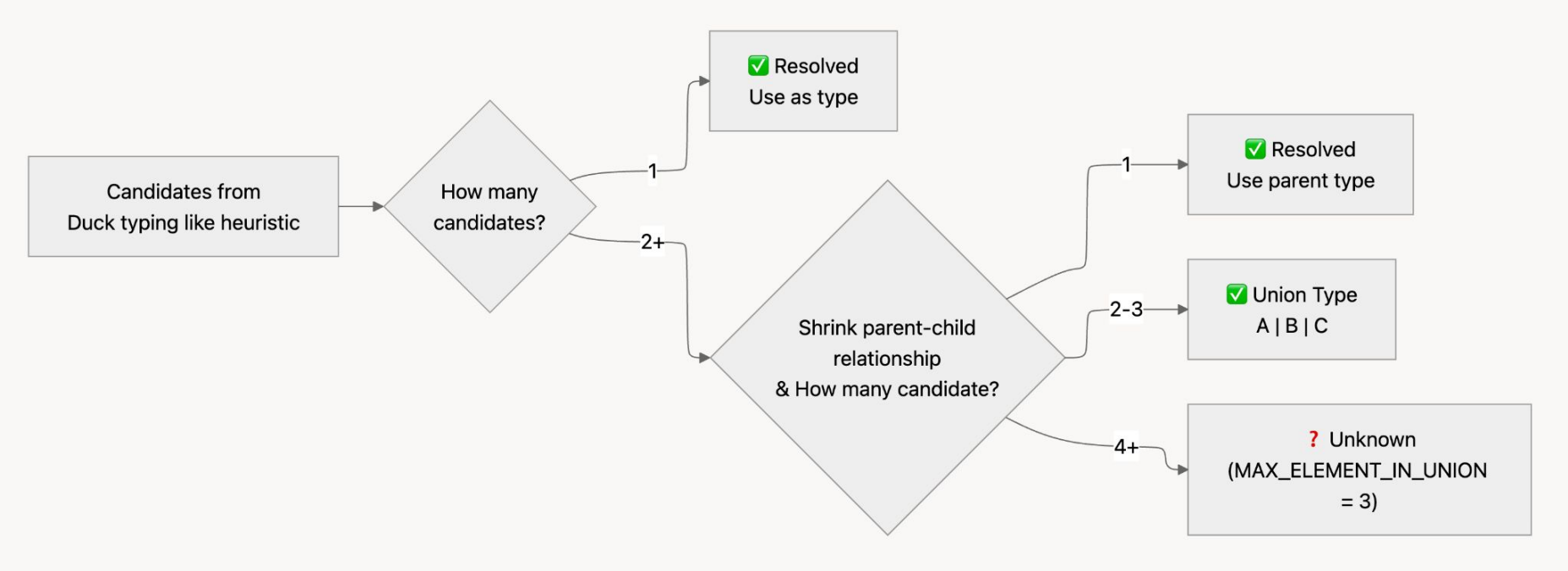


Inference strategy

Basic inference flow



Duck typing like heuristic flow



Candidate decision - Resolved as parent class

```ruby
class Recipe
  def comments = []
  def ingredients = []
end

class MacaronRecipe < Recipe
  def content = ""
  def tags = []
end

def check(recipe)
  recipe.comments
  recipe.ingredients
end
```

Guessed Signature: `(Recipe recipe) -> []`

[TypeGuessr Debug]

Reason: def check returns Recipe#ingredients (project)

Source: project

Candidate decision - Resolved as Union

```ruby
class RecipeA
  def comments = []
  def ingredients = []
end

class RecipeB
  def comments = []
  def ingredients = []
end

def check2(recipe)
  recipe.ingredients
  recipe.comments
end
```

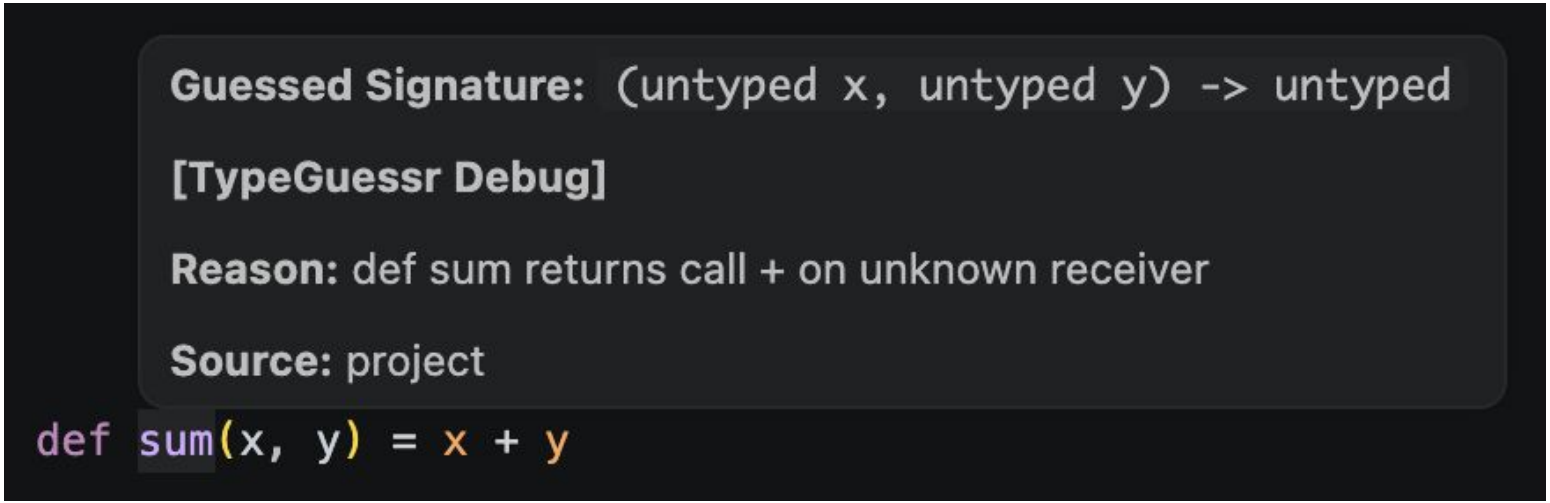
Guessed Signature: `(RecipeA | RecipeB recipe) -> untyped`

[TypeGuessr Debug]

Reason: def check2 returns call comments on unknown receiver

Source: project

Candidate decision - Ambiguous case



Exception: ivar

```ruby
class Recipe1
  attr_reader :name

  def initialize(name)
    @name = name
  end
end
recipe = Recipe1.new("Makaron")
recipe.name
```

```
name()

Guessed receiver: Recipe1

Learn more about guessed types

Definitions: a.rb

Guessed Signature: () -> untyped

[TypeGuessr Debug]

Reason: def name returns assigned from parameter without type info

Source: project
```

Exception: ivar

```ruby
class Recipe2
  attr_reader :name

  def initialize(name)
    @name = name
  end
  def grapheme_clusters_of_name
    @name.grapheme_clusters
  end
end
recipe = Recipe2.new("Makaron")
recipe.name
```

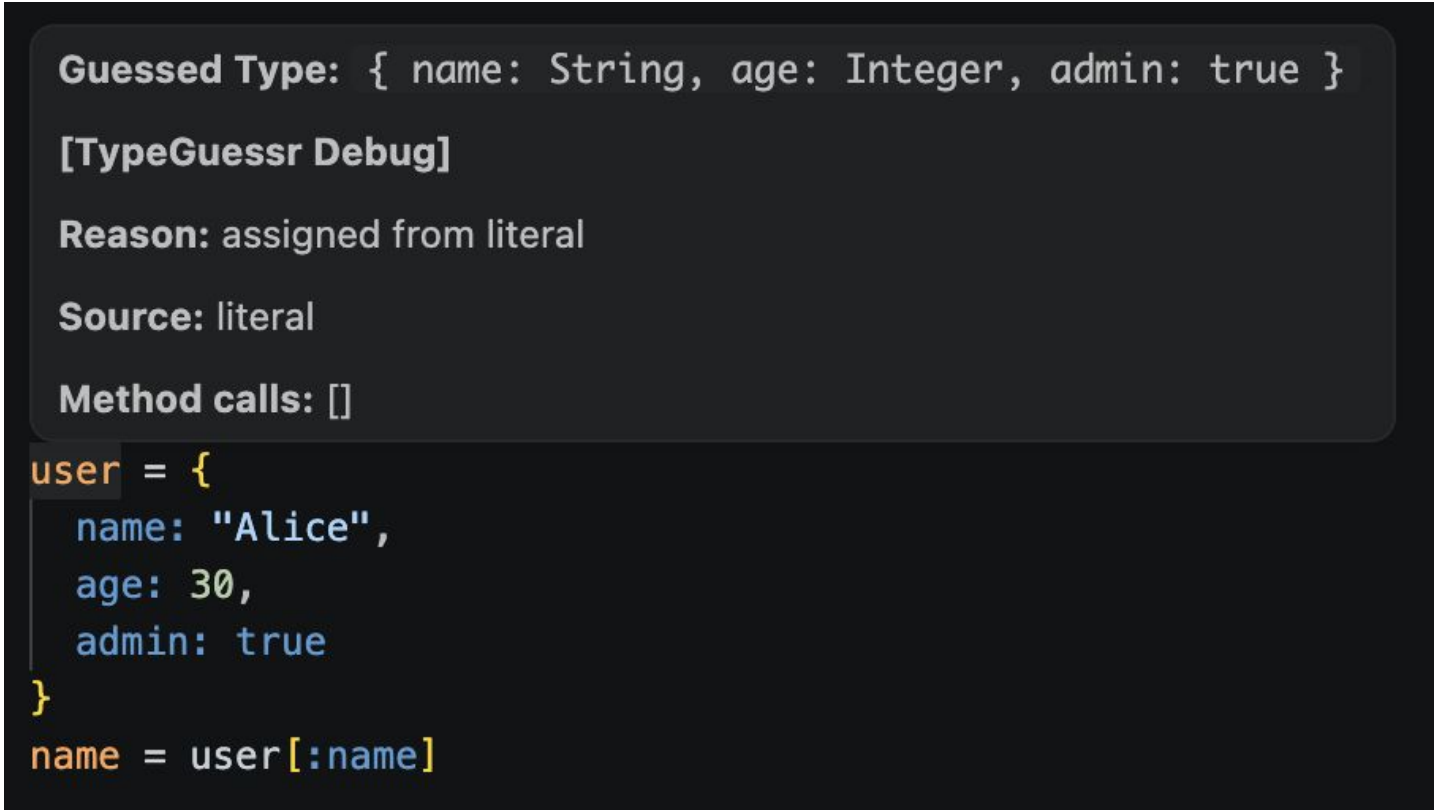
```
name()

Guessed receiver: Recipe2

Learn more about guessed types

Definitions: a.rb

Guessed Signature: () -> String

[TypeGuessr Debug]

Reason: def name returns assigned from parameter inferred from grapheme_clusters

Source: project
```

Expressiveness of the type

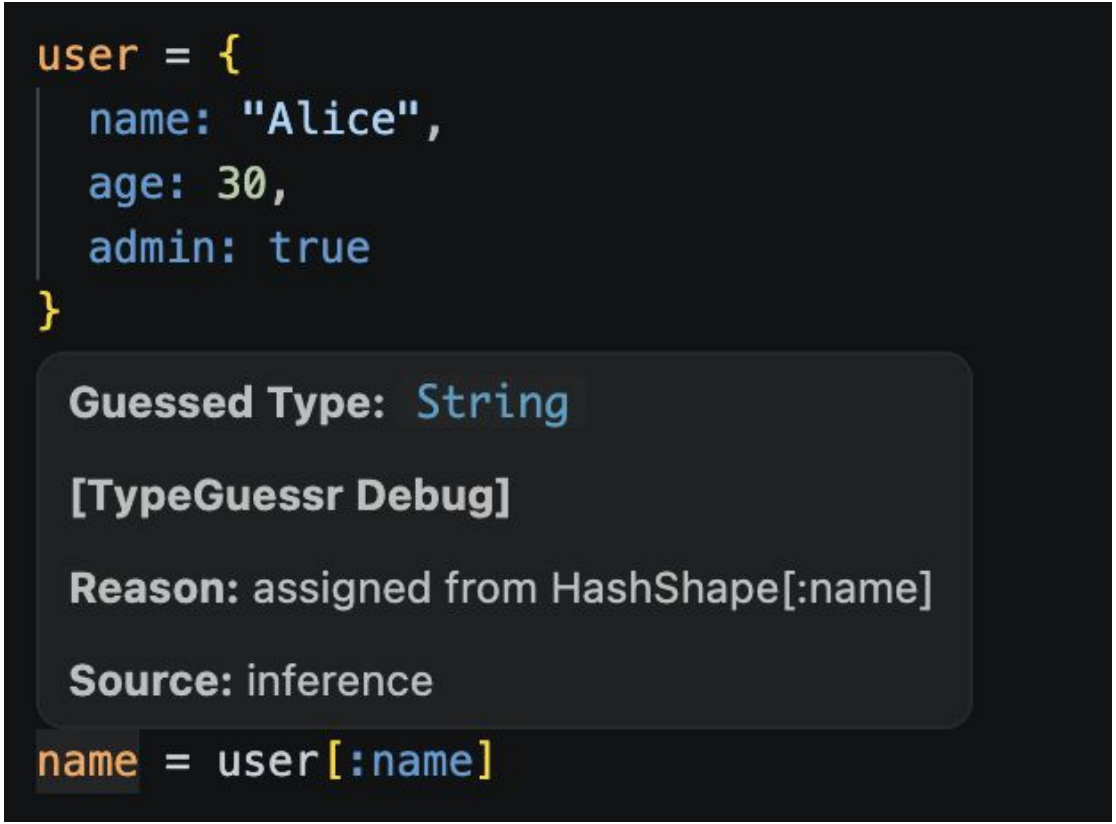
Structural type...?

- Record type
- Tuple type

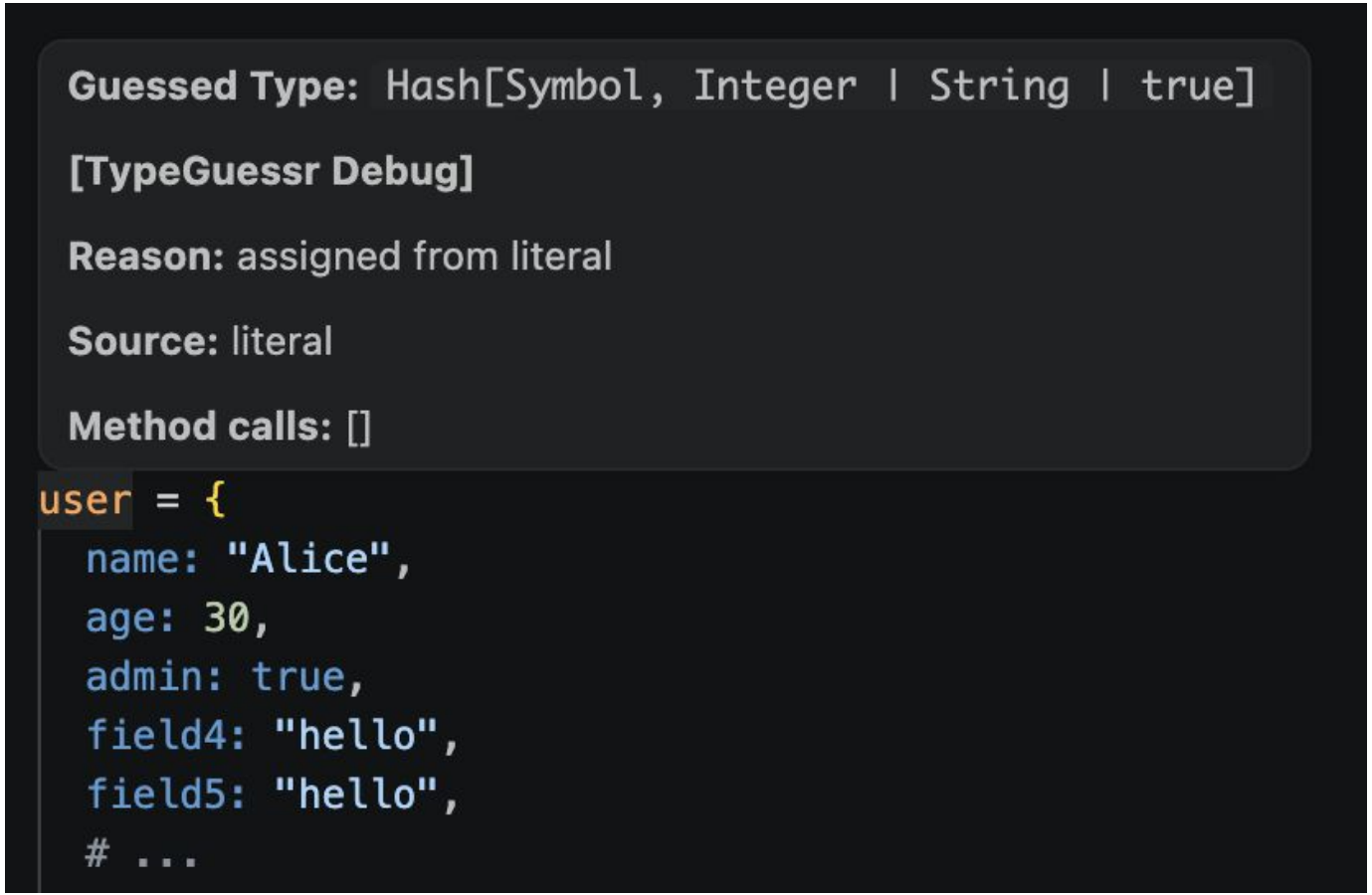
Record type - Hash



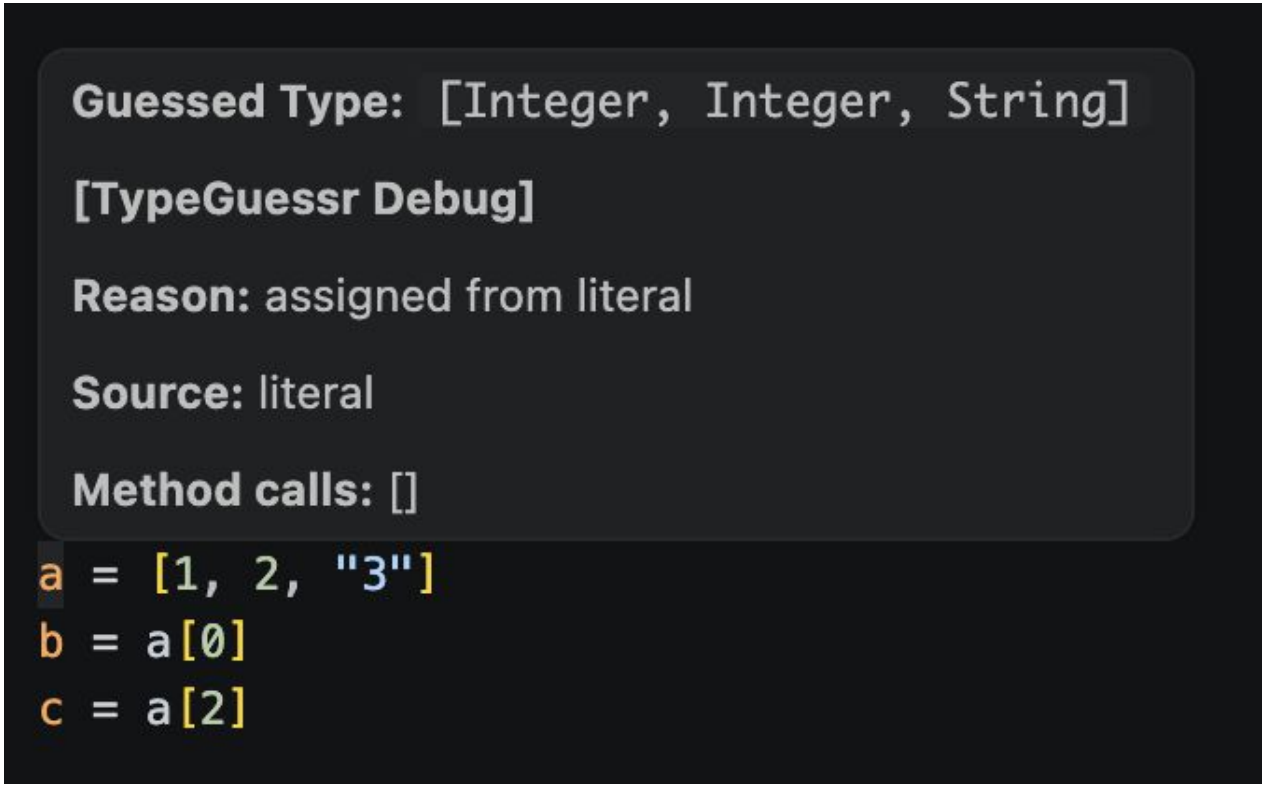
Record type - Hash



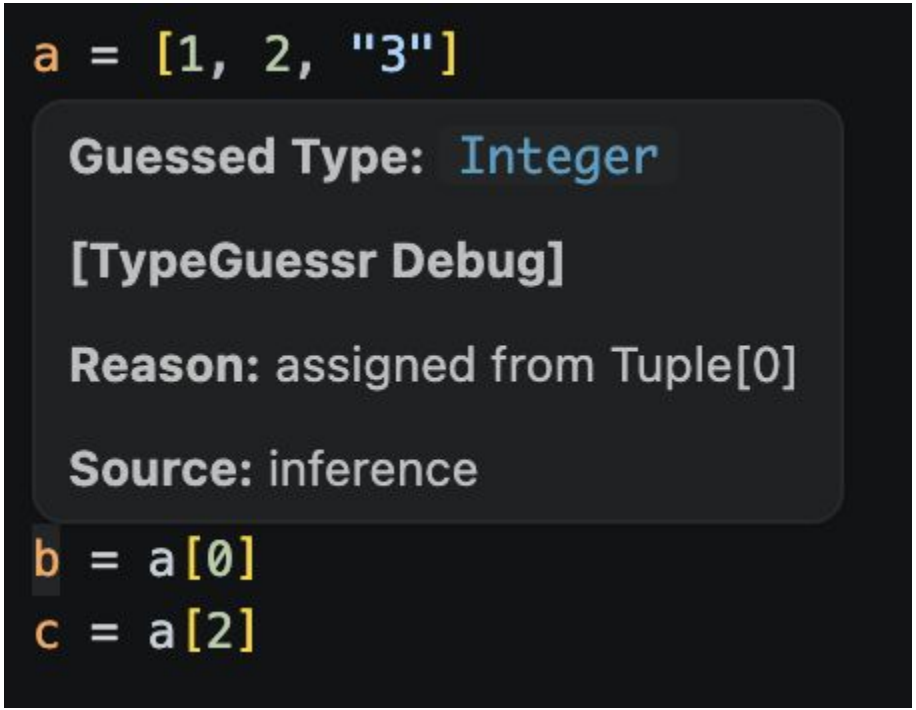
Record type - Hash

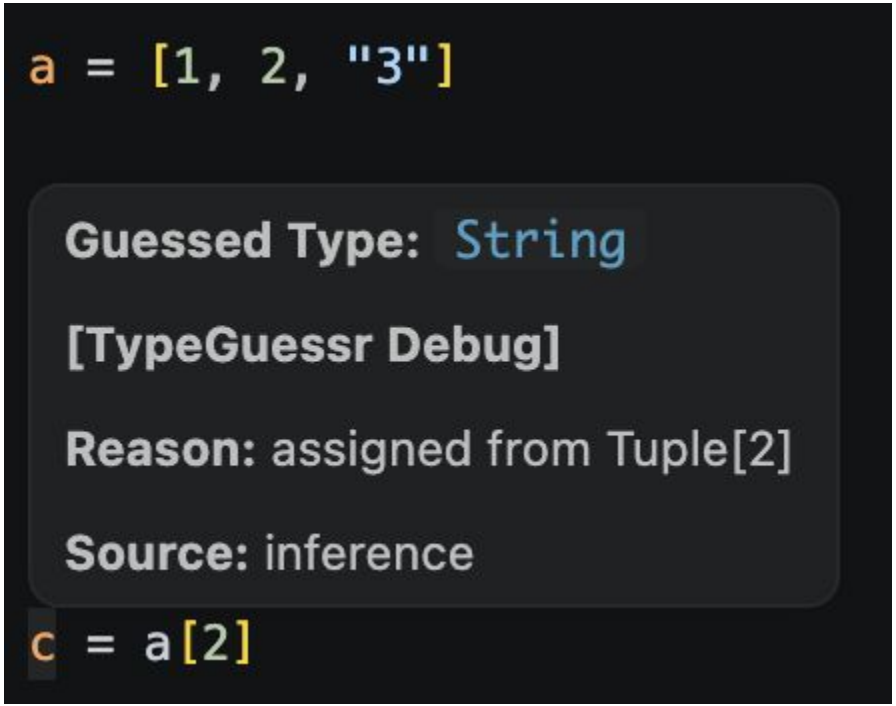


Tuple type - Array

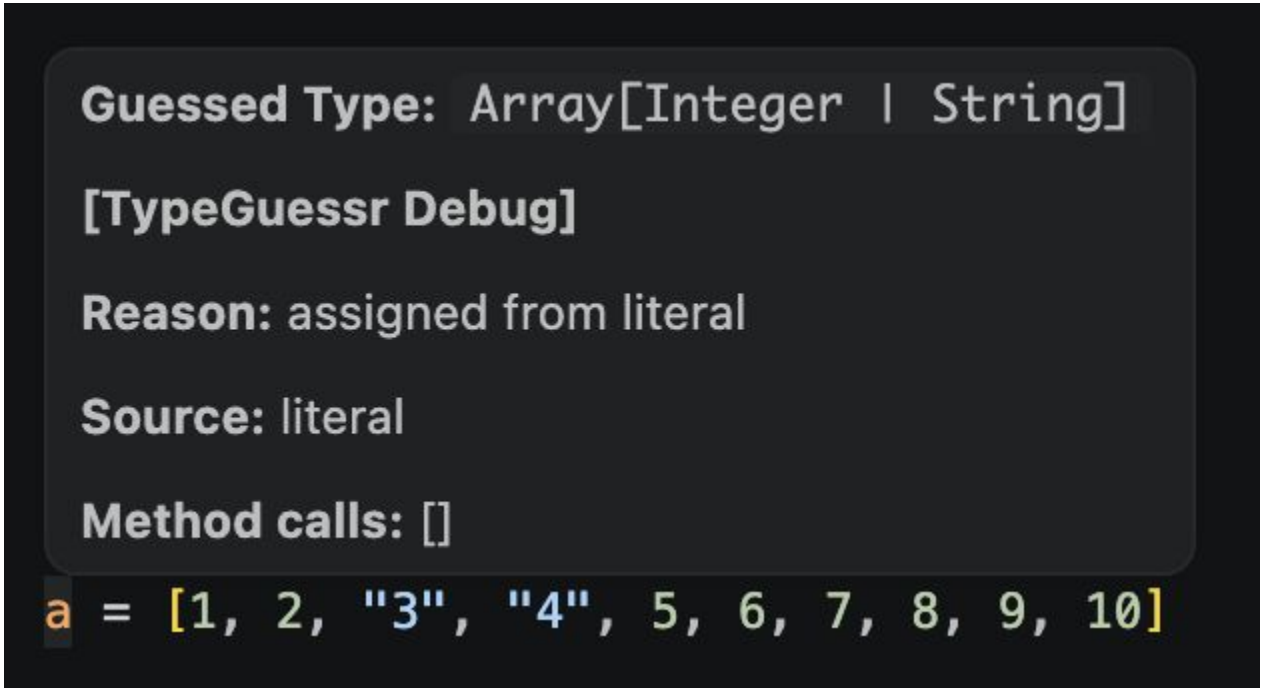


Tuple type - Array





Tuple type - Array



DSL support

DSL support… is hard

Everyone knows DSL support is hard for type inference on Ruby. Let’s see the patterns which makes inference hard with activerecord, well known gem to you.

Correctly tracking include / extend is hard

```ruby
module M
  extend ActiveSupport::Concern

  included do
    scope :disabled, -> { where(disabled: true) }
  end

  class_methods do
    ...
  end
end
```

Model attributes are unknown to begin with

```ruby
class User < ApplicationRecord; end
user = User.new
user.name # does it exist?
```

```ruby
User.select(:name, :email) # ????
```

How do existing tools handle this?

rbs_rails and tapioca, ruby-lsp-rails do essentially the same thing:

- Run project, and get the needed information at runtime
- Write out concrete type information to RBS based on that
- Or fetch type info when queried

type-guessr's Approach

type-guessr doesn't want users to write types, and ideally doesn't want to add RBS to the codebase.

- Decide to embed ActiveRecord-aware logic internally (experimental)
  - Trust ActiveRecord's interface stability for now
- Eventually we want a registration API to externalize this — future work.

Current implementation

For ActiveRecord, it has 2 parts:

- Queries the Rails runtime
  - column info, associations, enums, scopes…
- Base on above, generate methods and register those to index

Current implementation

```
user.name              # => String?
user.email             # => String?
user.age               # => Integer?
user.active            # => bool?
user.score             # => Float?
user.balance           # => BigDecimal?
user.bio               # => String?
```

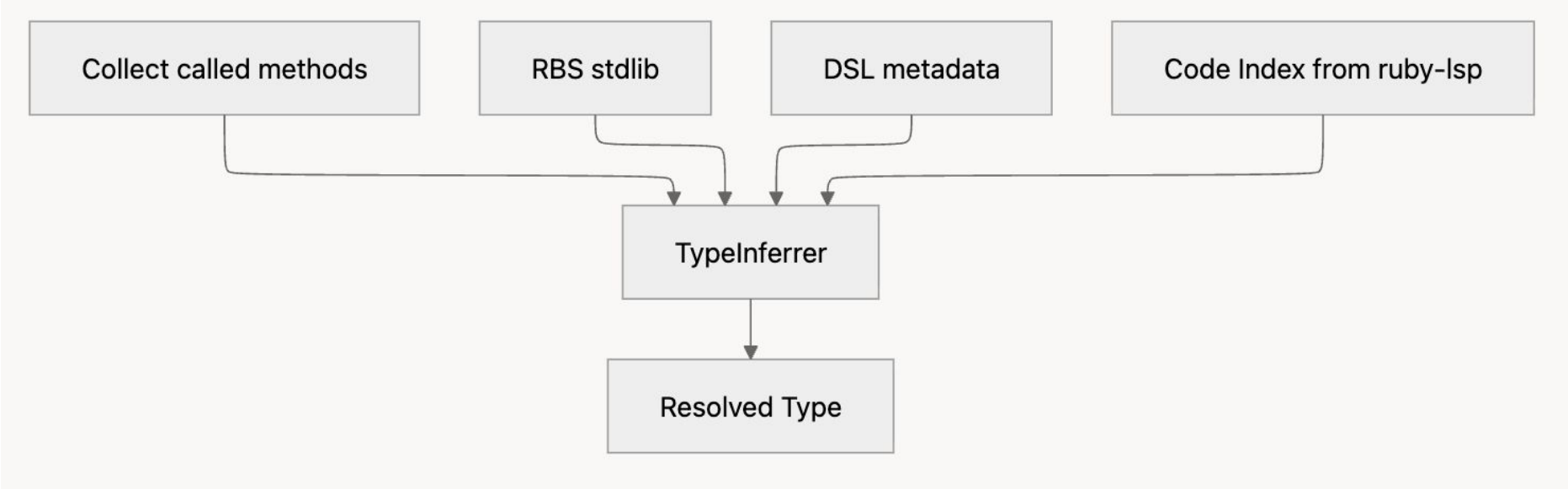
Current implementation

Design of type-guessr

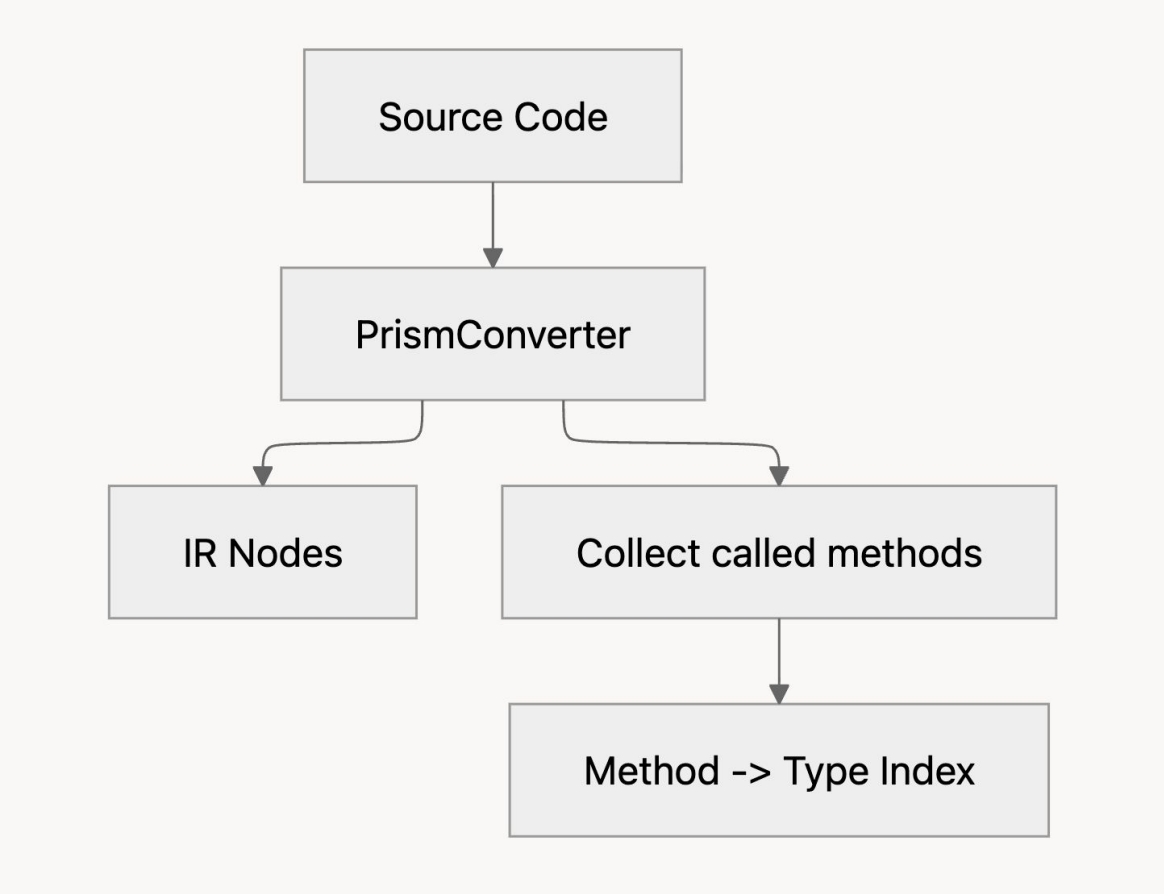
Impl as ruby-lsp add on

- To concentrate on inference logic, not LSP server implementation
  - Just providing type info, ruby-lsp will handle text on hover, candidate on autocompletion, def jump…
- Get methods generated by DSL from other add-on, such as ruby-lsp-rails

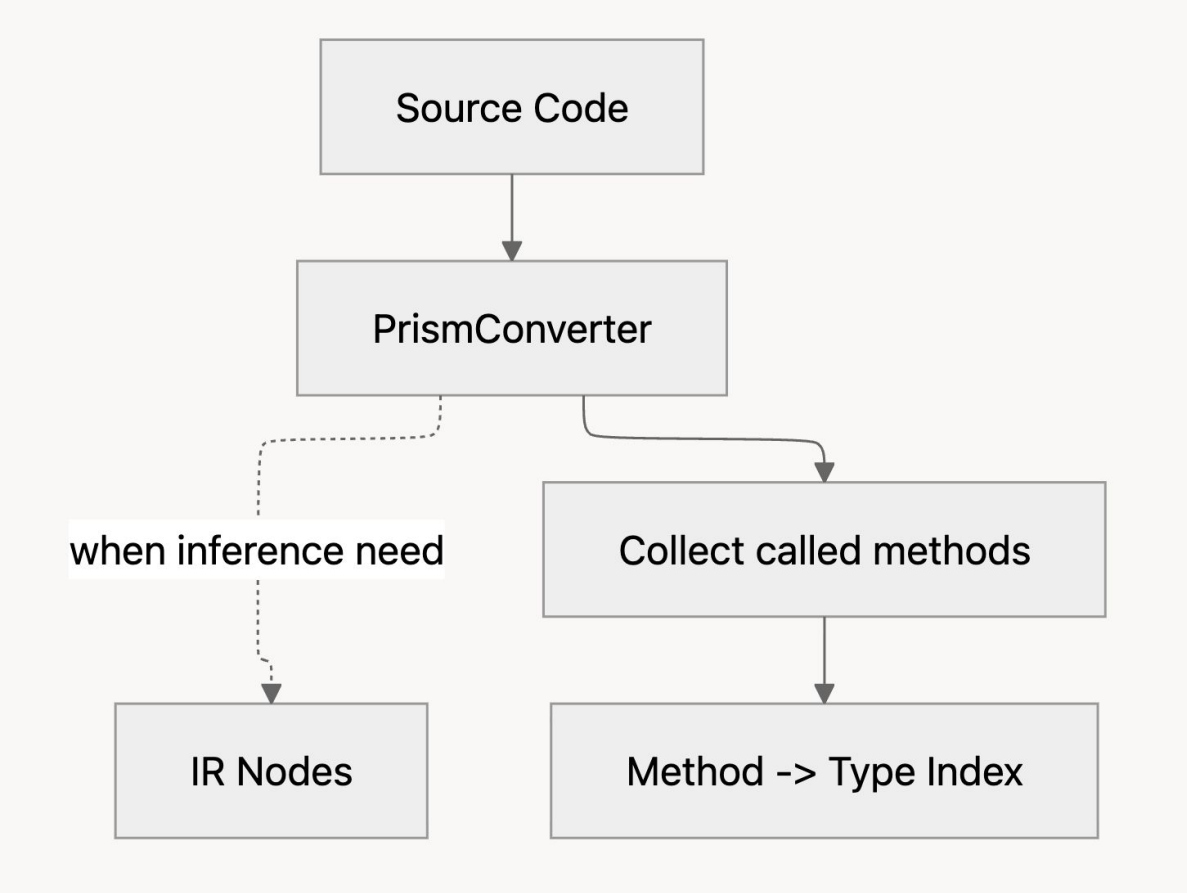
Type information sources



Index & type var graph building (project scope)



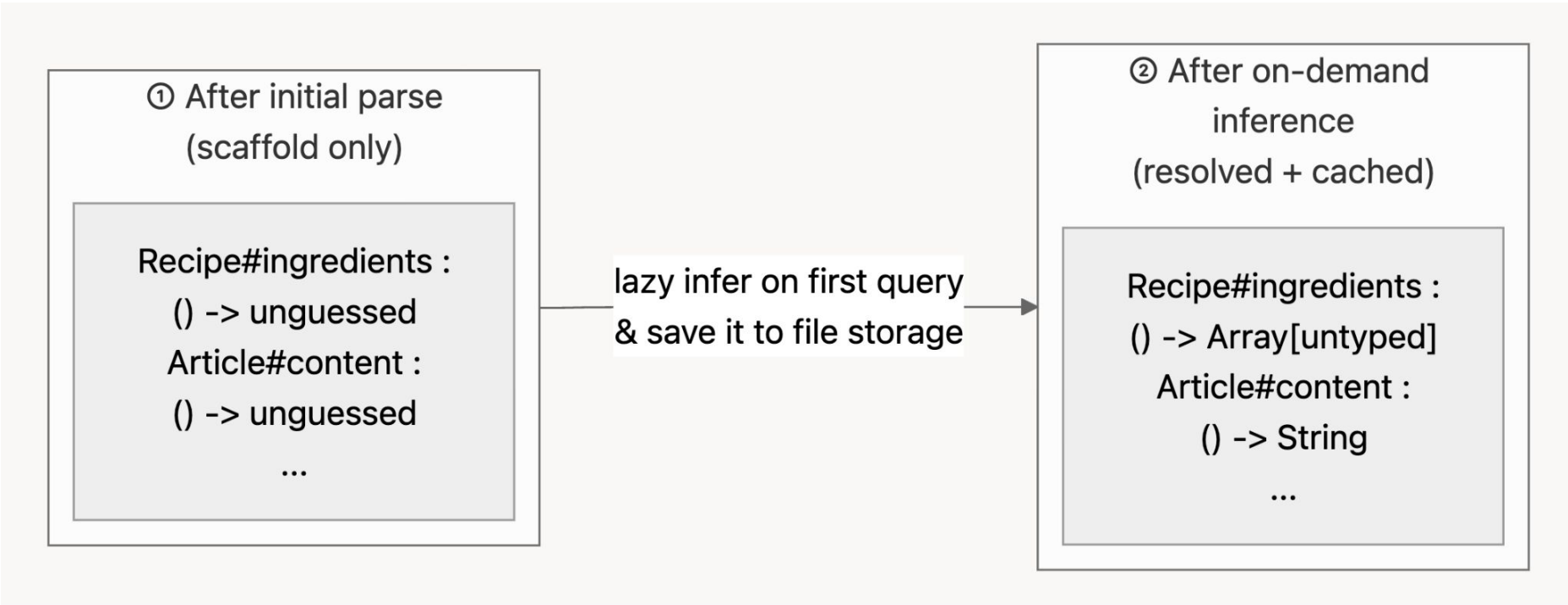
Index & type var graph building (in gem scope)



Lazy Inference

- Don’t infer at index time
- Do inference when requested with cache
- some gems are very slow to infer/parse as they have too many files. so with gems:
  - Collect constant and its method on indexing time.
  - Building IR nodes, inferring lazy.

Inference for gem



Numbers and Limitations

TypeGuessr Coverage Report

Generated: 2026-04-24 07:30:11

Summary

Metric	Value
Files Analyzed	3832
Node Coverage	79.6% (1770909/2223899)
Inference Coverage	40.1% (303594/756584)
Signature Score	0.39
Project Methods	48449

> **Node Coverage** includes trivially-typed nodes (LiteralNode, SelfNode). **Inference Coverage** excludes them to reflect actual type inference capability.

Speed

- For our company's Rails apps: startup under 20s with ruby-lsp. Inference under 2s without cache. Under 1s when warm.
- "An LSP must not interfere with editor operations"
  - Most inference done in 100-200ms
  - It's only occasionally slow…

Note: Benchmarks and coverage reports are in the repo

Limitation

- Weak against small methods
- Getting the correct method list is critical, otherwise everything falls into untyped
- Diagnostic is hard
  - Information to validate and information to aid inference are tangled together — still figuring this out
  - First step might be just to give warning when there is no matched constants?

Result

- ✅ Built "type-guessr" to show possibility of proposed heuristic.
  - fast enough
  - coverage is not enough, now
- ⏳ Extensive DSL support
- ⏳ Diagnostics
- ⏳ Adapt to AI age..?

Acknowledgement

The Ruby Association grant is the reason this got built, instead of staying on my "someday" list.

I met regularly with my mentor, Endoh-san, to talk through how the type inference should work.